The Embedded I/O Company

TEWS TECHNOLOGIES

TIP700-SW-42

VxWorks Device Driver

Digital Output 24V DC

Version 2.0.x

User Manual

Issue 2.0.2

October 2010

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com www.tews.com

TIP700-SW-42

VxWorks Device Driver

Digital Output 24V DC

Supported Modules:
TIP700

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©1996-2010 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	June 3, 1996
1.1	New Installation Chapter	April 28, 1998
1.2	Smaller Changes and Corrections	April 2, 2002
2.0.0	New Driver Interface Description	July 27, 2007
2.0.1	Carrier Driver description added	June 24, 2008
2.0.2	General revision	October 1, 2010

Table of Contents

1 INTRODUCTION 4
1.1 Device Driver 4
1.2 IPAC Carrier Driver 5
2 INSTALLATION 6
2.1 Include device driver in VxWorks projects 6
2.2 System resource requirement 6
3 I/O SYSTEM FUNCTIONS 7
3.1 tip700Drv() 7
3.2 tip700DevCreate() 9
4 I/O FUNCTIONS 11
4.1 open() 11
4.2 close() 13
4.3 ioctl() 15
4.3.1 TIP700_READ 17
4.3.2 TIP700_WRITE 18
4.3.3 TIP700_WDENABLE 19
4.3.4 TIP700_WDDISABLE 20
4.3.5 TIP700_WDTRIGGER 21

1 Introduction

1.1 Device Driver

The TIP700-SW-42 VxWorks device driver software allows the operation of the TIP700 IPAC module conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open(), close()* and *ioctl()* functions.

The TIP700-SW-42 device driver supports the following features:

- Write output value
- Read back the contents of the output
- Enabling, disabling and triggering the output watchdog
- Support for legacy and VxBus IPAC carrier driver
- SMP Support

The TIP700-SW-42 supports the modules listed below:

TIP700-10	16 isolated digital outputs 24V DC	IndustryPack
TIP700-20	8 isolated digital outputs 24V DC	IndustryPack

To get more information about the features and use of TIP700 devices it is recommended to read the manuals listed below.

TIP700 User manual
TIP700 Engineering Manual
CARRIER-SW-42 IPAC Carrier User Manual

1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so called Carrier Driver that hides all differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-42 is part of this TIP700-SW-42 distribution. It is located in directory CARRIER-SW-42 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-42 User Manual for a detailed description how to install and setup the CARRIER-SW-42 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP700-SW-42':

tip700drv.c	TIP700 device driver source
tip700def.h	TIP700 driver include file
tip700.h	TIP700 include file for driver and application
tip700exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP700-SW-42-2.0.2.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include device driver in VxWorks projects

For including the TIP700-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path. (For example: ./TIP700)

(2) Add the device drivers C-files to your project.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.2 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	---	---

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tip700Drv()

NAME

tip700Drv() - installs the TIP700 driver in the I/O system

SYNOPSIS

#include “tip700.h”

STATUS tip700Drv(void)

DESCRIPTION

This function initializes the TIP700 driver and installs it in the I/O system.

A call to this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tip700.h”

/*-------------------
  Initialize Driver
  -------------------*/
status = tip700Drv();
if (status == ERROR)
{
     /* Error handling */
}
```

RETURNS

OK, or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tip700DevCreate()

NAME

tip700DevCreate() – Add a TIP700 device to the VxWorks system

SYNOPSIS

```
#include “tip700.h”

STATUS tip700DevCreate
(
        char        *name,
        int         devIdx,
        int         funcType,
        void        *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the device to add to the system. The driver searches for supported devices in the following order: TIP700-10, TIP700-20.

Example: A system with 2x TIP700-10 and 1x TIP700-20 will assign the following device indices:

Module	Device Index
TIP700-10 (1st)	0
TIP700-10 (2nd)	1
TIP700-20	2

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter is unused and should be set to NULL.

EXAMPLE

```
#include “tip700.h”

STATUS            result;

/*-------------------------------------------------------
  Create the device "/tip700/0" for the first TIP700 module
  -------------------------------------------------------*/
result = tip700DevCreate("/tip700/0", 0, 0, NULL);
if (result == OK)
{
     /* Device successfully created */
}
else
{
     /* Error occurred when creating the device */
}
```

RETURNS

OK, or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_ioLib_NO_DRIVER	The driver has not been started.
EINVAL	Invalid input argument
EISCONN	The device has already been created
ENXIO	Desired module instance not found

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4 I/O Functions

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
        const char  *name,
        int         flags,
        int         mode
)
```

DESCRIPTION

Before I/O can be performed to the TIP700 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened, the name specified in tip700DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

/*----------------------------------------
  Open the device named "/tip700/0" for I/O
  ----------------------------------------*/
fd = open("/tip700/0", 0, 0);
if (fd == ERROR)
{
     /* handle error */
}
```

RETURNS

A device descriptor number, or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
STATUS close
(
     int          fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int       fd;
STATUS    retval;

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == ERROR)
{
     /* handle error */
}
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

```
#include “tip700.h”

int ioctl
(
        int     fd,
        int     request,
        int     arg
)
```

DESCRIPTION

Special I/O operation that does not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	Description
TIP700_READ	Read content of output register
TIP700_WRITE	Write data to output register
TIP700_WDENABLE	Enable Watchdog
TIP700_WDDISABLE	Disable Watchdog
TIP700_WDTRIGGER	Trigger Watchdog

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

Function dependent value (described with the function) or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.3.1 TIP700_READ

This I/O control function reads back the current content of the output register. The function specific control parameter **arg** is a pointer on an *unsigned short* value.

EXAMPLE

```
#include “tip700.h”

int                fd;
unsigned short     usValue;
int                retval;

/*-------------------------------------
  Read current output value
  -------------------------------------*/
retval = ioctl(fd, TIP700_READ, (int)&usValue);
if (retval != ERROR)
{
     /* function succeeded */
     printf("Value: 0x%04X\n", usValue);
}
else
{
     /* handle the error */
}
```

4.3.2 TIP700_WRITE

This I/O control function writes a value to the output register. For TIP700-10 modules, 16bit are valid. For TIP700-20 modules, only the lower 8bit will be used. The function specific control parameter **arg** is a pointer on an *unsigned short* value.

EXAMPLE

```
#include “tip700.h”

int                fd;
unsigned short     usValue;
int                retval;

/*-------------------------------------
  Write value to output register
  -------------------------------------*/
usValue = 0x1234;

retval = ioctl(fd, TIP700_WRITE, (int)&usValue);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.3.3 TIP700_WDENABLE

This I/O control function enables the watchdog.

The function specific control parameter **arg** is not used.

Note that the watchdog must be triggered at least every 120ms, otherwise all outputs will be reset to 0. A write access also triggers the watchdog.

EXAMPLE

```
#include “tip700.h”

int                fd;
int                retval;

/*-----------------------
  Enable Watchdog
  -----------------------*/
retval = ioctl(fd, TIP700_WDENABLE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.3.4 TIP700_WDDISABLE

This I/O control function disables the watchdog.

The function specific control parameter **arg** is not used.

EXAMPLE

```
#include “tip700.h”

int                fd;
int                retval;

/*-----------------------
  Disable Watchdog
  -----------------------*/
retval = ioctl(fd, TIP700_WDDISABLE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.3.5 TIP700_WDTRIGGER

This I/O control function triggers the watchdog. The function reads the current output value, and writes it back to the output register, which triggers the watchdog. If the watchdog is enabled, call this function at least every 120ms. Otherwise the output register will be reset to 0.

The function specific control parameter **arg** is not used.

EXAMPLE

```
#include “tip700.h”

int                fd;
int                retval;

/*-----------------------
  Trigger Watchdog
  -----------------------*/
retval = ioctl(fd, TIP700_WDTRIGGER, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```